2. Tecnología de Soporte para la Arquitectura

2.1 Lenguaje C++

C++ es un lenguaje de programación de propósito general desarrollado por Bjarne Stroustrup en la década de los 80's. Las características más importantes que ofrece este lenguaje son:

- Soporte para la programación orientada a objetos.
- Portabilidad.
- Brevedad.
- Programación modular.
- Compatibilidad con el lenguaje de programación C.
- Velocidad.

De acuerdo a Stroustrop (1997, p. 7), el criterio de diseño más importante de C++ fue la simplicidad pero también lo es que haya sido desarrollado teniendo como base C lo cual permite que exista una correspondencia muy cercana entre sus tipos de datos, operadores, sentencias y la forma en que la computadora trata los números, caracteres y direcciones.

Adicionalmente a las características antes mencionadas brinda a los programadores otra gran ventaja, y es que cuenta con una decena de implementaciones independientes, centenares de bibliotecas, libros de texto y manuales brindando un gran soporte a la comunidad de desarrolladores.

Es importante recalcar que C++ no fue diseñado para el cómputo numérico aunque hoy en día es una opción muy popular para el desarrollo de aplicaciones en las que el manejo de gráficos y las interfaces de usuario son una parte importante, por lo que cuenta con un gran respaldo en este ámbito y es una de las principales ventajas por las que se optó usar este lenguaje de programación.

2.2 STL de C++

La Standard Template Library (STL) es una colección de estructuras de datos contenedoras[1], algoritmos e iteradores que están escritos en C++. Cabe señalar que la STL no es la primera implementación de este tipo, ya que anteriormente la mayor parte de los compiladores disponían de bibliotecas similares o estaban disponibles varías bibliotecas comerciales, aunque su principal desventaja es que no eran compatibles entre sí, lo que suponía para los programadores la obligación de aprender las nuevas bibliotecas para migrar de un proyecto a otro o bien de uno a otro compilador.

Sin embargo, la STL fue adoptada por el comité ANSI de estandarización de C++, lo que significó que actualmente todos los compiladores lo tuvieran como una extensión más del lenguaje.

La STL provee algunas de los tipos de contenedores más generales y útiles permitiendo al programador seleccionar una estructura que mejor se ajuste a las necesidades de su aplicación con la certeza de que cualquiera de ellas está respaldada por una gran flexibilidad, eficiencia y bases teóricas.

[1] Una estructura de datos se dice que es contenedora si puede contener instancias de otras estructuras de datos.

Resumen de los contenedores estándar	
vector<T>	Un vector de tamaño variable.
list<T>	Una lista doblemente enlazada.
queue<T>	Una cola.
stack<T>	Una pila.
deque<T>	Una cola de doble punta.
priority_queue<T>	Una cola ordenada por valor.
set<T>	Un conjunto,
multiset<T>	Un conjunto en el cual un valor puede repetirse varias veces.
map<key,val>	Un arreglo asociativo.
multimap<key,val>	Un mapa en el cual una llave puede repetirse varias veces.

Figura 1. Contenedores estándar más importantes que ofrece la STL.

De acuerdo a las características de los contenedores estos se pueden dividir en tres categorías:

- Contenedores lineales. Son aquellos que almacenan los objetos de forma secuencial, permitiendo además el acceso a los mismos de forma secuencia y/o aleatoria.
- Contenedores asociativos. Son aquellos que almacenan los objetos asociándolos a una clave. Su principal ventaja es la rapidez con la que se pueden almacenar o recuperar los objetos del contenedor.
- Contenedores adaptados. Permiten cambiar un contenedor en un nuevo contenedor modificando la interface del primero.

Al hacer uso de la STL se obtienen varias ventajas, de entre las cuales podemos mencionar las siguientes:

- Al ser estándar, está disponible para todos los compiladores y plataformas, lo cual permite usar la misma biblioteca en todos los proyectos.

- Al ser una biblioteca de componentes reutilizables incrementa la productividad y reduce el tiempo de desarrollo.
- El desarrollo de las aplicaciones se hacen rápidamente ya que se construyen a partir de algoritmos eficientes.
- Proporciona su propia gestión de memoria, de tal forma que el programador podrá ignorar problemas tales como las limitaciones de la memoria del PC.

2.3 Interfaz de Programación de Aplicaciones OpenGL

OpenGL es un ambiente para desarrollo portable de aplicaciones gráficas interactivas 2D y 3D. Fue introducido en 1992 y desde entonces se ha convertido en la interfaz de programación de aplicaciones o API (por sus siglas en inglés) más utilizada a nivel mundial. OpenGL promueve la innovación y el rápido desarrollo de aplicaciones incorporando una amplia colección de herramientas de renderizado, mapeo de texturas, efectos especiales y muchas más funciones de visualización, por lo que es una API gráfica ideal sobre la cual basar nuestra arquitectura.

Rendimiento y Calidad Visual

Cualquier aplicación gráfica que requiera un rendimiento superior, de animación 3D a CAD[2] o hasta simulación visual puede explotar las capacidades de alta calidad que ofrece OpenGL. Estas capacidades permiten a los desarrolladores de diversas áreas como CAD/CAM[3]/CAE[4], entretenimiento, visualización médica y realidad virtual, producir y desplegar gráficos 2D y 3D convincentes.

[2] Computer-Aided Design que se refiere al diseño asistido por computadora.
[3] Computer-Aided Manufacturing que se refiere a la manufactura de productos asistida por computadora.
[4] Computer-Aided Engineering que se refiere a la ingeniería asistida por computadora.

Ventajas para el desarrollador

a. Estándar en la industria

La junta de revisión de la arquitectura (ARB por sus siglas en inglés) de OpenGL, es la encargada de llevar su especificación además de que es el único estándar gráfico multiplataforma y abierto.

b. Estable

Las implementaciones de OpenGL han estado disponibles por más de 7 años en una gran variedad de plataformas. Adicionalmente, la especificación lleva un buen control que permite anunciar actualizaciones en tiempo para que los desarrolladores adopten estos cambios. La retro compatibilidad asegura que las aplicaciones existentes no se vuelvan obsoletas.

c. Confiable y Portable

Todas las aplicaciones producen resultados consistentes para cualquier hardware compatible con el API de OpenGL, sin importar el sistema operativo o el sistema de ventanas.

d. Constante evolución

Debido a su diseño cuidadoso y vanguardista, OpenGL permite que la innovación en hardware sea accesible a través del mecanismo de extensiones de su API. De esta forma, las innovaciones aparecen de manera oportuna permitiendo a los desarrolladores y a los vendedores de hardware incorporar las nuevas características en sus ciclos normales de publicación.

e. Escalable

Las aplicaciones basadas en el API de OpenGL se ejecutan en sistemas desde PCs, estaciones de trabajo hasta supercomputadoras. Como resultado, las aplicaciones son escalables hacia cualquier tipo de máquina que elija el desarrollador.

f. Facilidad de uso

OpenGL está bien estructurado con un diseño intuitivo y comandos lógicos. Las rutinas eficientes de OpenGL generalmente resultan en aplicaciones con menos líneas de código que con otras bibliotecas o paquetes gráficos. Adicionalmente, los controladores de OpenGL encapsulan la información acerca del hardware subyacente liberando al desarrollador de tener la necesidad de diseñar para un hardware específico.

g. Documentado

Existe un gran número de libros publicados acerca de OpenGL junto con muchos ejemplos de código disponibles haciendo que la información acerca de OpenGL no sea costosa y sea fácil de obtener.

Pipeline o tubería de programación de OpenGL

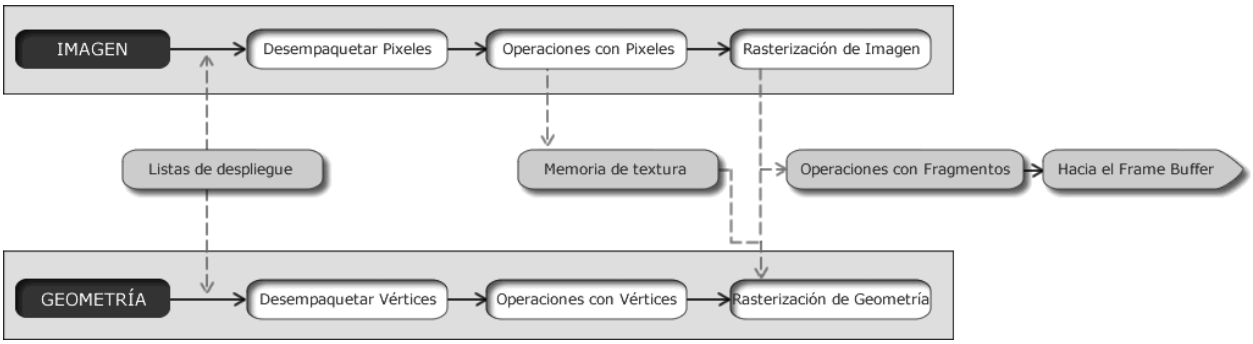

Figura 2. OpenGL opera sobre datos de imagen así como sobre primitivas geométricas

Simplificación de desarrollo de software

Las rutinas de OpenGL simplifican el desarrollo de aplicaciones gráficas, desde dibujar una simple geometría de un punto, línea, polígono relleno, hasta la creación de mapas de iluminación y texturizado más complejos. OpenGL da acceso a los desarrolladores a primitivas de imagen y geometría, listas de despliegue, transformaciones de modelo, iluminación y texturizado, anti-aliasing[5], blending[6] y otras más características.

[5] Anti-aliasing son técnicas que permiten obtener imágenes con bordes suavizados.

El estándar de OpenGL tiene relación con los lenguajes C, C++, Fortran, Ada y Java. Todas las implementaciones licenciadas de OpenGL vienen de una especificación única que es requerida para pasar un conjunto de pruebas de compatibilidad. Las aplicaciones que hacen uso de las funciones de OpenGL son fácilmente portables a un amplio rango de plataformas para maximizar la productividad del programador y reducir el tiempo de comercialización.

Todos los elementos del estado de OpenGL, inclusive los contenidos de memoria de textura y frame buffer, pueden ser obtenidos por una aplicación basada en OpenGL. También soporta aplicaciones de visualización con imágenes 2D tratadas como primitivas que pueden ser manipuladas como cualquier geometría 3D. Como se muestra en la Figura 2, las imágenes y los vértices que definen primitivas geométricas son pasadas a través del pipeline de OpenGL hacia el frame buffer.

La base para APIs avanzadas

Las empresas líderes en desarrollo de software utilizan OpenGL, junto con sus bibliotecas robustas de dibujado, como la base para APIs para gráficos 2D y 3D de alto nivel. Es por tal motivo que muchos de los desarrolladores recurren a las capacidades de OpenGL para entregar sus soluciones al mercado.

[6] El blending consiste en la mezcla de 2 colores para formar uno, permitiendo efectos como transparencia, filtros aditivos y más.

2.4 Shaders con GLSL

Los shaders son programas que permiten calcular efectos de renderizado directamente en el hardware gráfico con un grado alto de flexibilidad. Los shaders sustituyen los estados correspondientes al procesamiento de vértices y/o fragmentos con áreas programables que pueden realizar menor, igual o mayor número de operaciones que la funcionalidad fija. Los shaders fueron diseñados para permitir a los programadores realizar descripciones del comportamiento en estos puntos y por ello es imprescindible hacer uso de ellos para mejorar la calidad visual de los ambientes virtuales que serán dibujados por nuestra arquitectura.

Existen diferentes tipos de shaders:

a. Shaders de vértice (Vertex Shaders)

Se aplican sobre cada vértice que procesa la GPU. Se pueden manipular propiedades como la posición, el color, la coordenada de textura, pero no se pueden crear nuevos vértices.

b. Shaders de geometría (Geometry Shaders)

Permiten añadir o quitar vértices de una malla. Se utilizan para añadir detalle a volúmenes de forma no tan costosa como si se procesara en la CPU.

c. Shaders de pixel (Fragment Shaders)

Procesan el color para cada pixel individual. Generalmente se utilizan para calcular iluminación, tonalidad de color o efectos como bump mapping[7].

[7] Bump mapping es una técnica que permite crear la ilusión de relieves a través del uso de una imagen difusa (textura) y un mapa de normales.

¿Por qué usar shaders?

Los shaders permiten describir efectos de manera más simple y eficientes que la funcionalidad fija de OpenGL, también se pueden obtener mejores resultados e inclusive realizar efectos que serían imposibles con la funcionalidad fija.

Algunos efectos que se pueden obtener:

- Materiales realistas: metal, madera, piedra, etc.
- Iluminación más realista: luz por área, sombras suaves, etc.
- Ambientes naturales: fuego, agua, cielo, nubes, etc.
- Materiales no realistas: lápiz, pluma, carbón, etc.
- Nuevos usos de memoria de textura.
- Procesamiento digital de imágenes
- Animación

GLSL

La reciente tendencia en el hardware de gráficos ha sido remplazar la funcionalidad fija con programación en áreas que han crecido de manera excepcionalmente compleja (por ejemplo procesamiento de vértices y procesamiento de fragmentos). El lenguaje de shaders de OpenGL (GLSL por sus siglas en inglés), fue diseñado para permitir a los programadores modificar el procesamiento que ocurre en ciertos puntos del pipeline de OpenGL.

El lenguaje GLSL está basado en ANSI C y muchas de los rasgos han permanecido a excepción de cuando entran en conflicto con el rendimiento o la facilidad de implementación. El lenguaje C ha sido extendido con tipos de dato como vectores y matrices para hacerlo más conciso con las operaciones típicas que son usada en gráficos 3D. Algunos otros mecanismos del lenguaje C++ fueron tomados, como la

sobrecarga de funciones basada en los tipos de dato de los argumentos, y la habilidad de declarar variables donde sea necesario en lugar de ser al inicio de los bloques de código.

Procesadores Programables

Existen 2 tipos de procesadores programables:

- Vértices
- Fragmentos (pixeles)

Cuando los procesadores están activos se integran al estado de OpenGL e inactivan la funcionalidad fija. Los procesadores programables ejecutan shaders de vértices y fragmentos respectivamente. Fueron diseñados para trabajar con OpenGL por lo que tienen salidas y entradas que convergen en el pipeline estándar.

Las aplicaciones pueden pasar datos a los shaders a través de atributos y variables *uniform* definidas por el usuario. Los shaders de vértices pueden comunicarse con el procesamiento subsecuente utilizando variables de salida especiales y variables *varying* integradas. Los shaders de fragmentos obtienen datos del procesamiento previo a través de variables de entrada especiales y variables *varying* integradas.

a. Procesador de vértices

Es una unidad programable que opera sobre los vértices de entrada y sus valores asociados (un solo vértice a la vez).

Está diseñado para realizar operaciones tradicionales como:

- Transformación de vértices
- Transformación y normalización de normales
- Generación de coordenadas de textura
- Cálculos de iluminación

- Aplicación de color de material

Cuando se habilitan los shaders de vértices la siguiente funcionalidad de OpenGL no se aplica:

- La matriz de vista de modelo no se aplica a los vértices entrantes
- La matriz de proyección no se aplica
- La matriz de texturas no se aplica
- Las normales no son transformadas a coordenadas de vista
- No se generan coordenadas de textura
- No se aplica iluminación por vértice
- No se aplica color de material
- No se aplica color indexado

Por lo tanto el shader de vértices debe implementar las operaciones sustituidas de la funcionalidad fija dependiendo de las necesidades de aplicación.

El shader de vértices no reemplaza la funcionalidad que requiere del conocimiento de más de un vértice:

- La división de perspectiva por coordenadas de corte
- Mapeo al puerto de vista
- Rango de profundidad
- Recorte
- Determinación de cara frontal
- Ajuste de color, textura, normales
- Procesamiento de color final

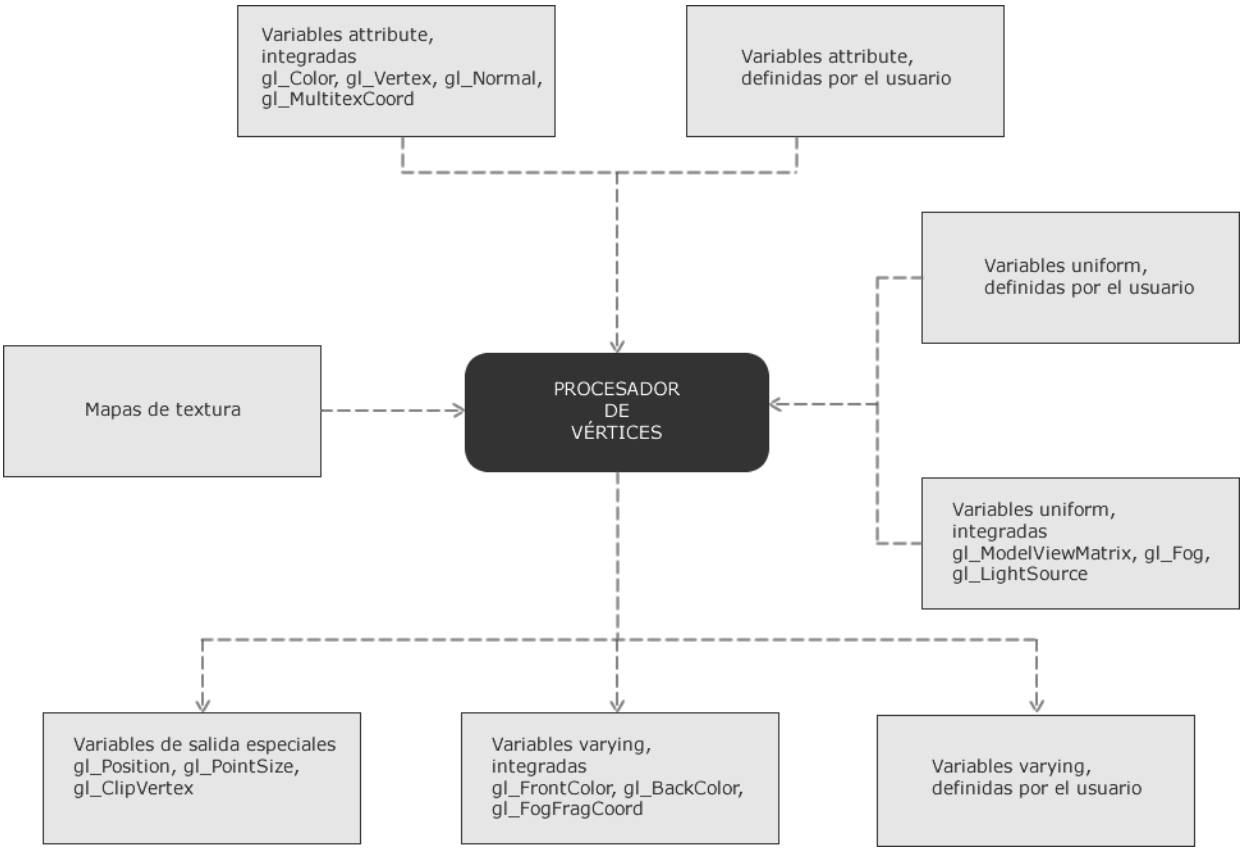

Figura 3. Procesador de vértices

b. Procesador de fragmentos

Es una unidad programable diseñada para operar fragmentos y sus datos asociados. Puede realizar las siguientes operaciones tradicionales:

- Operaciones en los valores interpolados de los vértices
- Acceso a texturas
- Aplicación de texturas
- Aplicación de niebla
- Suma de color

Ejecutan shaders de fragmentos y no pueden cambiar la posición (x, y) de un fragmento.

Cuando se habilitan los shaders de fragmentos la siguiente funcionalidad de OpenGL no se aplica:

- El ambiente y funciones de textura no son aplicados
- No se mapean las texturas
- No se realiza la suma de color
- No se aplica la niebla

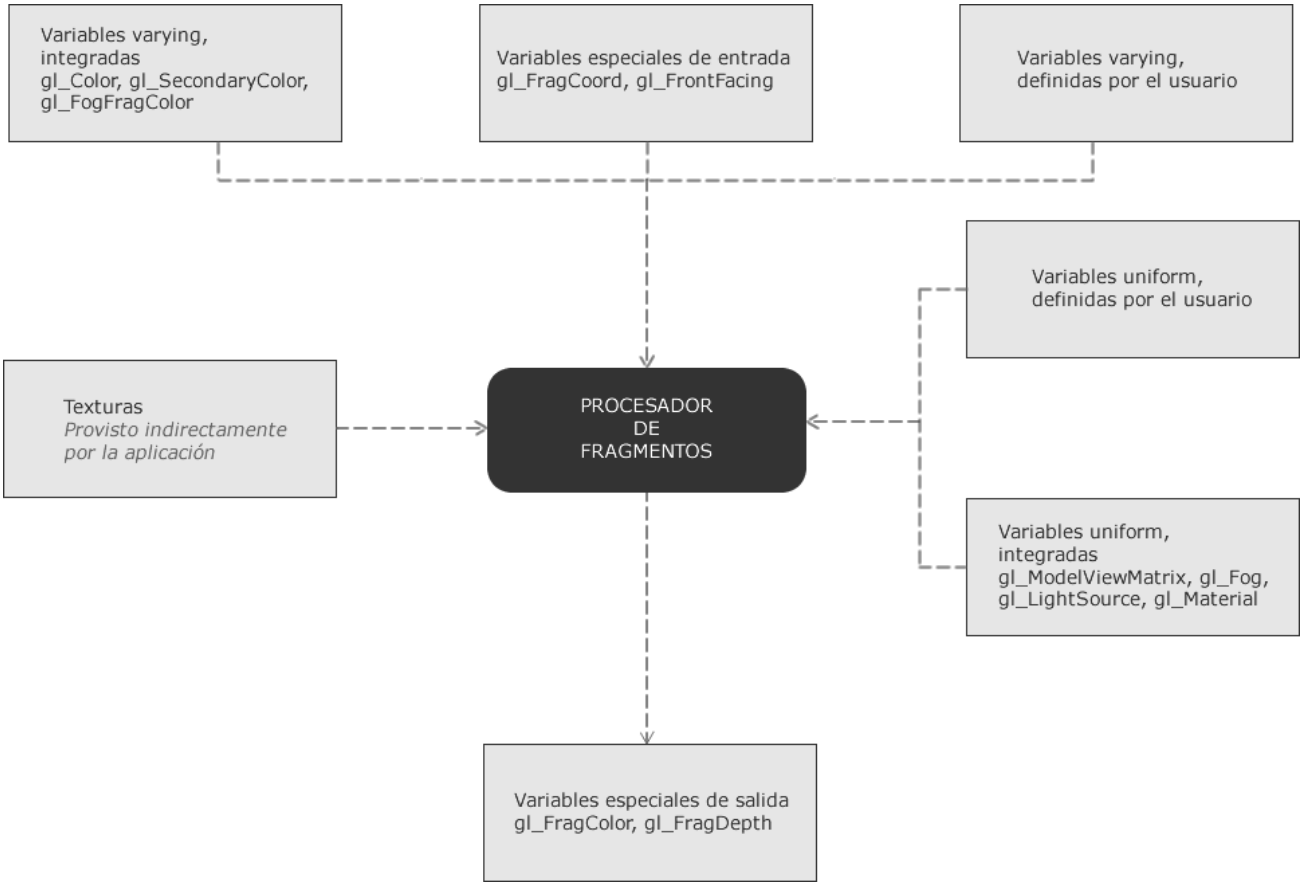

Figura 4. Procesador de fragmentos

2.5 Animación con Cal3D

Cal3D es una biblioteca de animación escrita en C++ con licencia LGPL[8], que es independiente tanto de la API gráfica como de la plataforma donde es utilizada.

Originalmente fue diseñada para ser usada como un cliente 3D del proyecto Worldforge[9] pero finalmente se convirtió en un proyecto independiente que puede ser usado en cualquier otro tipo de proyectos. Al momento en que el proyecto dio inicio, la tecnología más prometedora y flexible para la animación de los personajes era la basada en esqueletos, la cual proporcionaba mucha libertad en el proceso de animación y fue elegida como el elemento central de la biblioteca Cal3D.

De acuerdo a los creadores de la biblioteca, los objetivos contemplados en el diseño de este proyecto fueron:

- Funcionalidad.
- Facilidad de uso.
- Portabilidad.
- Escalabilidad.
- Flexibilidad.
- Rendimiento.
- Interoperabilidad.

Las características más relevantes que fueron consideradas para elegir usar esta biblioteca dentro de nuestro proyecto fueron las siguientes:

[8] En inglés, Lesser General Public License, permite hacer uso del código fuente de manera libre sin necesidad de pagar por él siendo así una licencia de software libre.
[9] Ver http://www.worldforge.org/ para más información

- Provee un sistema de control que permite manipular la secuencia y la mezcla de las animaciones, además de que le permite soportar diferentes tipos de ellas, tales como ciclos, acciones o poses.
- Cuenta con un manejo flexible de los materiales y las mallas lo que permitiría cambiar completamente la apariencia de un modelo.
- El diseño de la biblioteca está orientado a que ésta sea simple, lógica e intuitiva.
- Dado que biblioteca esta codificada en C++ y únicamente depende de la STL, puede ser usada en diferentes plataformas pudiendo utilizar tanto OpenGL como DirectX sin ningún problema, ya que no lleva a cabo el manejo del renderizado y la administración del texturizado por sí misma.
- Debido a que su diseño estuvo contemplado para formar parte de un cliente MMORPG[10] (Massively Multiplayer Online Role-Playing Game) el rendimiento global para manejar varios modelos al mismo tiempo es muy importante, lo cual se logró al optimizar los cálculos dentro la propia biblioteca.

Cal3D puede ser visto como dos piezas: la biblioteca de C++ y el exportador. El exportador como tal nos permite crear los archivos de los modelos elaborados bajo el formato Cal3D, el cual es el único que puede ser cargado por la biblioteca. Actualmente el software de modelado que tiene soporte para el exportador de Cal3D es 3ds Max y Maya. Por otra parte, la biblioteca de Cal3D provee a los desarrolladores la interfaz para la manipulación de las animaciones.

10 Son videojuegos de rol que permiten a miles de jugadores introducirse en un mundo virtual de forma simultánea a través de la Internet e interactuar entre ellos.

2.6 Audio con FMOD Ex

El sistema de sonido FMOD Ex es un motor de audio creado por Firelight Technologies para impulsar los límites creativos de la implementación de audio en videojuegos y similares, usando los mínimos recursos y manteniendo la escalabilidad. FMOD Ex está pensado para el uso de desarrolladores de videojuegos, desarrolladores multimedia, diseñadores de audio, músicos e ingenieros de audio. Es importante mencionar que se utilizó éste motor bajo la licencia no comercial, la cual establece que el producto derivado no estará destinado a fines comerciales y que tampoco se usará la biblioteca de FMOD para su venta o distribución comercial. Es por ello que las características provistas por FMOD Ex es ideal para enriquecer con audio los escenarios desplegados por nuestra arquitectura.

Arquitectura de procesamiento digital de señales (DSP)

La arquitectura para mezclar señales de FMOD utiliza cálculos de punto flotante con interpolaciones completas de 32 bits para proveer la máxima calidad de audio en la suma de señales. Utiliza una arquitectura basada en nodos que permite una flexibilidad de acceso, sub mezcla y salida en canales de elección para el programador. Los canales de entrada pueden ser mapeados para cualquier canal de salida mediante una simple matriz de dos dimensiones.

API de FMOD Ex

Una de las ventajas de este motor de audio es su integración con el lenguaje C++ que permite acceder a su funcionalidad a través de clases (programación orientada a objetos) como la clase del sistema, de sonido, de canal, de DSP y más.

2.7 Manejo de archivos XML con TinyXML

TinyXML es un parser XML escrito originalmente en C++ por Lee Thomason que puede ser fácilmente integrado dentro de otros programas bajo la licencia zlib, la cual permite que sea usado tanto en código abierto como en código lucrativo.

La funcionalidad que TinyXML provee para la manipulación de archivos XML es básica y consiste en parsear un documento XML y construir a partir de él un Modelo de Objetos del Documento (DOM[11]) que puede ser leído, modificado y guardado. Dada la simplicidad y facilidad de uso, TinyXML se adapta a nuestros requerimientos para carga y lectura de archivos de configuración usados por nuestra arquitectura.

Se debe tener en cuenta que este parser tiene ciertas limitaciones, entre las que podemos enumerar:

- No usa ni procesa entidades DTD[12] o XLS[13].
- En términos de codificación, TinyXML da soporte a UTF-8, permitiendo manipular archivos XML en cualquier lenguaje. Además soporta “Legacy Mode”, la codificación usada antes de UTF-8 y que se podría describir mejor como “ASCII extendido”.

La ventaja de usar el formato XML es que su formato está bien estructurado, permitiendo crear cualquier clase de etiquetas en el documento. De esta forma todos aquellos formatos de archivos creados para almacenar datos de una aplicación pueden ser totalmente reemplazados con XML y usar un único parser para todos ellos.

[11] En inglés, Document Object Model
[12] Ver, Document Type Definition
[13] Ver, Extensible Stylesheet Language

Para finalizar, es importante mencionar que no se aplicó ningún tipo de encriptación en los archivos XML, ya que son usados como archivos de configuración para el usuario y no supone ningún riesgo para el funcionamiento de la arquitectura.